- Assignment 2 is due Wednesday AM! Please post on Piazza.

- Course project preference sheet: Out today!
  - Propose your own?
  - Or rank 1 through 5 of some project ideas we have

- Group!
LIMITATIONS OF DATA PARALLEL

- Overhead increases as num GPUs increases.

- Overhead is lower for <=8 GPUs, higher after that.

- Backward pass All Reduce is very high for some model types.

- Communication overhead (fraction of training time spent in communication stalls).

8xV100s with NVLink (AWS) PyTorch + NCCL 2.4

“fraction of training time spent in communication stalls”
MODEL PARALLEL TRAINING

- **5 layers in model**
- **Low utilization**
- **Limited memory req. are lower & can use heterogeneous machines**
- **Comm is constant as num workers increases.**
**Key Idea:**
Instead of 1 input batch, feed in "k" batches

- Handle scenarios where some stages are slow.

**Advantages?**
- Fixed comm. overhead
- High utilization

- Partitioning
- Scheduling of FWD, BCKWD
- Learning

**Steady State util is high**

**Pipeline Parallel**

- **Worker 1**
  - Forward Pass: 1, 2, 3, 4
  - Backward Pass: 1, 1, 5

- **Worker 2**
  - Forward Pass: 1, 2, 3, 4
  - Backward Pass: 2, 2, 5

- **Worker 3**
  - Forward Pass: 1, 2, 3, 4
  - Backward Pass: 3, 3, 5

- **Worker 4**
  - Forward Pass: 1, 1, 1, 2
  - Backward Pass: 4, 4, 4

- **Scheduling**
  - **Startup State**
    - Fixed overhead partitioning
  - **Steady State**
    - High utilization

- **Time**
  - Forward Pass: 1, 2, 3, 4
  - Backward Pass: 1, 1, 5
CHALLENGE 1: WORK PARTITIONING

Goal: Balanced stages in the pipeline. Why?
Steady state throughput is the throughput of the slowest stage

Stages can be replicated! Ex: Two stage pipeline, but first stage is replicated
WORK PARITIONING

Profiler: computation time for forward, backward for each layer
size of output activations, gradients (network transfer)
size of parameters (memory)

Dynamic programming algorithm
Intuition: Find optimal partitions within a server,
Then find best split across servers using that

*Given*
- model
- cluster
  L GPUs, network

Static planning
CHALLENGE 2: WORK SCHEDULING

Traditional data parallel

forward iter(i)
backward iter(i)
forward iter(i+1)
...

Pipeline parallel: Worker can
Forward pass to push to downstream
Backward pass to push to upstream

Either run forward pass next batch or backward pass for prior batch
**CHALLENGE 2: WORK SCHEDULING**

Num active batches $\approx \frac{\text{num\_workers}}{\text{num\_replicas\_input}}$

Schedule one-forward-one-backward (1F1B) – Worker 3

Round-robin for replicated stages $\rightarrow$ Worker 2

same worker for fwd, backward

> makes sure pipeline is making progress

---

The diagram shows a schedule for the workers over time, indicating different stages of the forward and backward passes, with times marked along the x-axis.

- Worker 1:
  - Replicated stages
  - Forward Pass (blue)
  - Backward Pass (green)
  - Idle (gray)

- Worker 2:
  - Replicated stages
  - Forward Pass (blue)
  - Backward Pass (green)
  - Idle (gray)

- Worker 3:
  - Forward Pass (blue)
  - Backward Pass (green)
  - Idle (gray)
CHALLENGE 3: EFFECTIVE LEARNING

Naïve pipelining
Different model versions forward and backward

Batch no. 5
- fwd pass uses model v2.
- bck pass used model v4.

Keep model v2
- stashed

batch 5
≡ v2
CHALLENGE 3: EFFECTIVE LEARNING

Weight stashing
Maintain multiple versions of the weights
One per active mini-batch

Use latest version for forward pass.
Retrieve for backward
No guarantees across stages!

\[
\text{batch no. 5} = [32 \text{ images}] \rightarrow m_{v1} \rightarrow \text{Activations} \rightarrow m_{v2} \rightarrow \text{Activations}
\]
STALENESS, MEMORY OVERHEAD

How to avoid staleness:

Vertical sync → in the first stage, record model version used

Memory overhead

Similar to data parallel?

→ lose this benefit with vertical sync.
Pipeline parallelism: Combine inter-batch and intra-batch
Partitioning: Replication, dynamic programming
Scheduling: IFIB
Weight management: Stashing, vertical sync
DISCUSSION

https://forms.gle/5cf16BWn6Dziey6e6
### List two takeaways from the following table for data parallelism

- **Data parallel** - not always bad
- Speedup higher

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Model Size</th>
<th>GPUs (#Servers x #GPUs/Server)</th>
<th>PipeDream Config</th>
<th>Speedup over DataParallel (Epoch Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50</td>
<td>97MB</td>
<td>4x4 2x8</td>
<td>16</td>
<td>1x 1x</td>
</tr>
<tr>
<td>VGG-16</td>
<td>528MB</td>
<td>4x4 2x8</td>
<td>15-1</td>
<td>5.28x 2.98x</td>
</tr>
<tr>
<td>GNMT-8</td>
<td>1.1GB</td>
<td>3x4 2x8</td>
<td>Straight 16</td>
<td>2.95x 1x</td>
</tr>
</tbody>
</table>
What are some other workload scenarios (e.g. things we discussed for MapReduce or Spark) that could use similar ideas of pipelined parallelism? Develop such one example and its execution.
NEXT STEPS

Next class: Parameter Server

Assignment 2 is due soon!

Course project preference form out today!